

# Research Ideology

28 October 2006

Alessio Guglielmi

University of Bath

<http://alessio.guglielmi.name>

Proof theory is one of the theoretical pillars of computer science. Originally intended to provide foundations for mathematics, it turned out to be a fundamental instrument for understanding computation. Despite the successes, proof theory is having difficulties in coping with the increasing demands of computer science. These difficulties stem from proof theory being too biased towards mathematical foundations. The purpose of my research is to overcome the limitations of present-day proof theory by adopting a new approach directly inspired by computer science.

One of the most celebrated successes of proof theory in computer science is known as the ‘Curry-Howard correspondence’ between normalisation in some deductive system and computation in some formalism like the lambda calculus. Analogously, logic programming and automated theorem proving find their natural foundations in proof theoretic formalisms like the sequent calculus and tableaux systems. In general, the design of declarative languages for computer science can only be done efficiently by having clear in mind the insights and technical achievements of proof theory.

In addition to providing a theoretical backbone for the syntax of languages, proof theory has also deep connections to their semantics. In fact, most of the modern developments in the semantics of computation happen in conjunction to proof theoretic investigations. Arguably, the most notable example is game semantics, which is now developed in close association with linear logic, which in turn is one of the most striking breakthroughs in proof theory, motivated by computation. The mutual interchange and dependence between proof theory and the theory of computation are more and more evident.

Game semantics and linear logic are motivated by the desire of understanding concurrent computation. This style of computation is profoundly different than the more traditional, non-concurrent, functional model, captured by the Turing machine and the lambda calculus. In the functional model computers compute functions, i.e., the emphasis of the computation is more on sequential algorithms to be performed in isolation than on distributed computation mostly based on asynchronous exchange of information, performed by parallel computing agents in a network.

The new style of computation is driven by the technological advances on networks, from the internet down to local networks and parallel, multiprocessor computers. The languages, verification and security tools needed to control these increasingly complex technologies push theoretical computer science, and proof theory, beyond the limits of what is reasonably achievable by traditional methods. The formalisms of proof theory were conceived for mathematics, which demands languages that are functional in nature: a mathematical proof is an isolated object, it is a static construction made on axioms, where no interaction or communication takes place between its components. A typical communication protocol is an inherently different object, which requires a different language and a different formalism to reason about that language.

The latest developments in proof theory clearly go in the right direction: linear logic and its offspring are, in many ways, languages designed with communication and distributed computation in mind, and they are indeed successful in capturing aspects of concurrent computation. However, all this only goes halfway through: while the languages of linear logic and its akin are adequate for concurrent computation, the proof theoretic formalisms used to reason about those language are not, because they are still those conceived for mathematics.

These formalisms, i.e., the sequent calculus, natural deduction and the newly conceived proof

nets (which are not deductive, though), all analyse a language by resorting to a certain meta-level formalism that still intimately is governed by classical logic. There is then a mismatch between the meta level, which is classical, and the object level, which behaves, for example, according to the algebra of linear logic connectives. This mismatch has serious technical consequences that range from unnecessary complexity to the impossibility of observing certain properties of computational interest. My recent research objective was to eliminate the mismatch between the object and meta level by employing the radical, new idea of *deep inference*. In addition to removing the mismatch, deep inference is inspired by such notions as locality and atomicity of communication, which are features of capital importance in distributed computation.

Deep inference achieves the result by using at the meta level the same object level language one is interested in studying. This induces a dramatic technical simplification in the way new languages can be defined, and provides for them several properties of direct computational relevance. However, on the proof theoretic side, very little of the old techniques can be used and much has to be redone from scratch, because deep inference challenges the very basic traditional tricks of the trade. After six years of intense development, we have now at our disposal several powerful techniques, which allow us to plan, with a high degree of confidence, for several years of future developments at all levels, from theoretical to practical.

We have already solved long standing, open problems, like for example giving a characterisation of process-algebras sequential composition in a deductive system, what linear logic could not achieve, precisely because of the inability of the sequent calculus to express the necessary form of noncommutativity. This result opens up new perspectives in very practical problems like the verification of security protocols. Such protocols can be described in a process algebra by using both sequential and parallel composition, and the problem of checking the protocol's safety can be formulated as a planning problem, for example via a logic programming language obtained from the process algebra. In order to do so, we need a deductive system, which is precisely what the old proof theory fails to deliver and what deep inference allows.

Moreover, deep inference brings into the picture several combinatorial properties which are provably impossible with traditional methods. These properties typically greatly enhance our ability of dealing with computations in a modular way. This will prove important both operationally, by allowing for more efficient search methods, and semantically, by inducing more refined semantics than currently possible. These two aspects mean more efficient computations and better ways of proving implementations correct.

On the theoretical side, deep inference revealed surprising regularities between disparate logic languages. A startling, recent observation has been the discovery that all of linear and classical logic, including all their fragments and several of their extensions, like modal logics, can be obtained by adopting one single, and simple!, rule scheme. From this we can conceive a great unification in proof theory and also a satisfying understanding of several proof theoretical phenomena that currently escape a unified understanding.

An outcome of this research will be sort of an automated process for generating languages endowed with a rich proof theory without having to hand tune them individually. Given the algebraic features one needs for the problem at hand (e.g., designing certain protocols), it will be possible to generate automatically a language, a compiler and a set of tools for reasoning about that language, something that currently might need years to a dedicated, expert proof theorist.

Like Robin Milner, I believe in the fruitfulness of trying to marry theory and practice. This is what I did with deep inference: I isolated some important features of modern computer science not adequately dealt with by proof theory, then I worked on them in a purely mathematical, abstract way. The results have largely exceeded my expectations: in fact, we made several totally unexpected discoveries, and we are now foreseeing many applications.